	Type	L #	Hits	Search Text	DBs	Time Stamp	Comments
1	BRS	L1	4	branch SAME fetch near3 instruction SAME third adj address	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/04 11:36	
2	BRS	L2	0	1 and Java adj bytecode	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/04 11:37	
3	BRS	L3	633	Java adj bytecode	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/04 11:37	
4	BRS	L4	19	3 and((jump or branch) SAME counter SAME address)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/04 11:38	
5	BRS	L5	19	3 and ((jump or branch) SAME counter SAME address)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/04 11:39	
6	BRS	L6	19	5 not 1	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/04 11:39	

Search Results

Search Results for: **[(branch or jump) and instruction and address and java and bytecode an count and second and third and timer]**
Found **15** of **84 searched out of 129,763.**

Search within Results

> Advanced Search > Search Help/Tips

Sort by: Title Publication Publication Date **Score** Binder

Results 1 - 15 of 15 short listing

1 Maté: a tiny virtual machine for sensor networks 82
Philip Levis , David Culler
Tenth international conference on architectural support for programming languages and operating systems on Proceedings of the 10th international conference on architectural support for programming languages and operating systems (ASPLOS-X) October 2002
Volume 37 , 30 , 36 Issue 10 , 5 , 5
Composed of tens of thousands of tiny devices with very limited resources ("motes"), sensor networks are subject to novel systems problems and constraints. The large number of motes in a sensor network means that there will often be some failing nodes; networks must be easy to repopulate. Often there is no feasible method to recharge motes, so energy is a precious resource. Once deployed, a network must be reprogrammable although physically unreachable, and this reprogramming can be a significan ...

2 Practicing JUDO: Java under dynamic optimizations 82
Michał Cierniak , Guei-Yuan Lueh , James M. Stichnoth
ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2000 conference on Programming language design and implementation May 2000
Volume 35 Issue 5
A high-performance implementation of a Java Virtual Machine (JVM) consists of efficient implementation of Just-In-Time (JIT) compilation, exception handling, synchronization mechanism, and garbage collection (GC). These components are tightly coupled to achieve high performance. In this paper, we present some static anddynamic techniques implemented in the JIT compilation and exception handling of the Microprocessor Research Lab Virtual Machine (MRL VM), ...

3 ASHs: application-specific handlers for high-performance messaging 80
Deborah A. Wallach , Dawson R. Engler , M. Frans Kaashoek
IEEE/ACM Transactions on Networking (TON) August 1997
Volume 5 Issue 4

4 Deadline analysis of interrupt-driven software 77
Dennis Brylow , Jens Palsberg
ACM SIGSOFT Software Engineering Notes , Proceedings of the 9th European software engineering conference held jointly with 10th ACM SIGSOFT international symposium on Foundations of software engineering September 2003

Volume 28 Issue 5

Real-time, reactive, and embedded systems are widely used throughout society (e.g., flight control, railway signaling, vehicle management, medical devices, and many others). For real-time, interrupt-driven software, timely interrupt handling is part of correctness. It is vital for software verification in such systems to check that all specified deadlines for interrupt handling will be met. Such verification is a daunting task because of the large number of different possible interrupt arrival s ...

5 Virtual machine monitors: Xen and the art of virtualization 77
Paul Barham , Boris Dragovic , Keir Fraser , Steven Hand , Tim Harris , Alex Ho , Rolf Neugebauer , Ian Pratt , Andrew Warfield
Proceedings of the nineteenth ACM symposium on Operating systems principles October 2003

Numerous systems have been designed which use virtualization to subdivide the ample resources of a modern computer. Some require specialized hardware, or cannot support commodity operating systems. Some target 100% binary compatibility at the expense of performance. Others sacrifice security or functionality for speed. Few offer resource isolation or performance guarantees; most provide only best-effort provisioning, risking denial of service.This paper presents Xen, an x86 virtual machine monit ...

6 Fast detection of communication patterns in distributed executions 77
Thomas Kunz , Michiel F. H. Seuren
Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research November 1997

Understanding distributed applications is a tedious and difficult task. Visualizations based on process-time diagrams are often used to obtain a better understanding of the execution of the application. The visualization tool we use is Poet, an event tracer developed at the University of Waterloo. However, these diagrams are often very complex and do not provide the user with the desired overview of the application. In our experience, such tools display repeated occurrences of non-trivial commun ...

7 Compilation and run-time systems: DELI: a new run-time control point 77
Giuseppe Desoli , Nikolay Mateev , Evelyn Duesterwald , Paolo Faraboschi , Joseph A. Fisher
Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture November 2002

The Dynamic Execution Layer Interface (DELI) offers the following unique capability: it provides fine-grain control over the execution of programs, by allowing its clients to observe and optionally manipulate every single instruction---at run time---just before it runs. DELI accomplishes this by opening up an interface to the layer between the execution of software and hardware. To avoid the slowdown, DELI caches a private copy of the executed code and always runs out of its own private cache.In ...

8 Compilation and run-time systems: Vacuum packing: extracting hardware-detected program phases for post-link optimization 77
Ronald D. Barnes , Erik M. Nystrom , Matthew C. Merten , Wen-mei W. Hwu
Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture November 2002

This paper presents Vacuum Packing, a new approach to profile-based program optimization. Instead of using traditional aggregate or summarized execution profile weights, this approach uses a transparent hardware profiler to automatically detect execution phases and record branch profile information for each new phase. The code extraction algorithm then produces code packages that are specially formed for their corresponding phases. The algorithm compensates for the incomplete and often incoheren ...

9 A survey of processors with explicit multithreading 77
Theo Ungerer , Borut Robič , Jurij Šilc
ACM Computing Surveys (CSUR) March 2003
Volume 35 Issue 1

Hardware multithreading is becoming a generally applied technique in the next generation of microprocessors. Several multithreaded processors are announced by industry or already into production in the areas of high-performance microprocessors, media, and network processors.A multithreaded processor is able to pursue two or more threads of control in parallel within the processor pipeline. The contexts of two or more threads of control are often stored in separate on-chip register sets. Unused i ...

10 Middleware performance analysis: Performance monitoring of java applications 77
M. Harkema , D. Quartel , B. M. M. Gijsen , R. D. van der Mei
Proceedings of the third international workshop on Software and performance July 2002
Over the past few years, Java has evolved into a mature platform for developing enterprise applications. A critical factor for the commercial success of these applications is end-to-end performance, e.g., in terms of response times, throughput and availability. This raises the need for the development, validation and analysis of performance models to predict performance metrics of interest. To develop and validate performance models, insight in the execution behavior of the application is essent ...

11 The benefits and costs of DyC's run-time optimizations 77
Brian Grant , Markus Mock , Matthai Philipose , Craig Chambers , Susan J. Eggers
ACM Transactions on Programming Languages and Systems (TOPLAS) September 2000
Volume 22 Issue 5
DyC selectively dynamically compiles programs during their execution, utilizing the run-time-computed values of variables and data structures to apply optimizations that are based on partial evaluation. The dynamic optimizations are preplanned at static compile time in order to reduce their run-time cost; we call this staging. DyC's staged optimizations include (1) an advanced binding-time analysis that supports polyvariant specialization (enabling both single-way and multi ...

12 Session summaries from the 17th symposium on operating systems principle (SOSP'99) 77
Jay Lepreau , Eric Eide
ACM SIGOPS Operating Systems Review April 2000
Volume 34 Issue 2

13 The 1999 ICFP programming contest 77
Norman Ramsey , Kevin Scott
ACM SIGPLAN Notices March 2000
Volume 35 Issue 3

14 Comparing mostly-copying and mark-sweep conservative collection 77
Frederick Smith , Greg Morrisett
ACM SIGPLAN Notices , Proceedings of the first international symposium on Memory management October 1998
Volume 34 Issue 3
Many high-level language compilers generate C code and then invoke a C compiler for code generation. To date, most, of these compilers link the resulting code against a conservative mark-sweep garbage collector in order to reclaim unused memory. We introduce a new collector, MCC, based on an extension of *mostly-copying collection*.We analyze the various design decisions made in MCC and provide a performance comparison to the most widely used conservative mark-sweep collector (the Boehm-Dem ...

15 Back to the future: the story of Squeak, a practical Smalltalk written in itself 77
Dan Ingalls , Ted Kaehler , John Maloney , Scott Wallace , Alan Kay
ACM SIGPLAN Notices , Proceedings of the 12th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications October 1997
Volume 32 Issue 10

Squeak is an open, highly-portable Smalltalk implementation whose virtual machine is written entirely in Smalltalk, making it easy to. debug, analyze, and change. To achieve practical performance, a translator produces an equivalent C program whose performance is comparable to commercial Smalltalks.Other noteworthy aspects of Squeak include: a compact object format that typically requires only a single word of overhead per object; a simple yet efficient incremental garbage collector for 32-bit d ...

Results 1 - 15 of 15 short listing